Justin Kan

My Entrepreneurship Story

HACKERMONTHLY

Issue 33 February 2013

Curator
Lim Cheng Soon

Contributors
Justin Kan
Rohin Dhar
Rob Spectre
Derek Sivers
Charles Leifer
Rob Pike
Bram Moolenaar
Clay Allsopp
Feross Aboukhadijeh
Joel Perras
Reginald Braithwaite
Jeff Atwood

Proofreaders
Emily Griffin
Sigmarie Soto

Printer
MagCloud

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com*, a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be "anything that gratifies one's intellectual curiosity." Every month, we select from the top voted articles on Hacker News and print them in magazine format. For more, visit *hackermonthly.com*

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

Published by
Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.

Cover: Justin Kan

Contents

FEATURES

Photo: *flickr.com/photos/jontintinjordan/5513995496/*

STARTUPS

SPECIAL

PROGRAMMING

For links to Hacker News dicussions, visit *hackermonthly.com/issue-33*

My Entrepreneurship Story

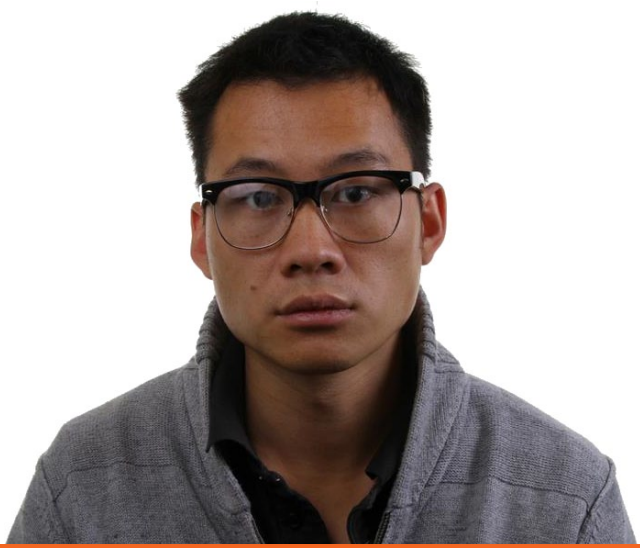
By JUSTIN KAN

My parents were entrepreneurs in the beginning. My mom had her own real estate agency and having that example was really, I think, a big part of it. There are a lot of people out there who would like to do something, but they don't because they're getting a nice paycheck.

For me, I tell myself all the time, "It doesn't matter." Those external things or how much money you have isn't really that important. The thing that's important is that you enjoy what you're doing every day or every hour. I think that's something most people don't get the opportunity to do. It's easy to make no money when you've never made any money. When I was at Yale as an undergraduate, I started this company called Kiko, which was a web calendar. Kiko was in the first class of Y Combinator Companies, which is a seed fund created by Paul Graham, who was investing in startups by younger entrepreneurs.

Y Combinator, at the time, was in a building in Cambridge. We went there for an interview and they told us they didn't like the idea of Kiko, but they thought we might be promising. We showed up, and I didn't say a single thing. I said, "Hey, I'm Justin." And they called us back that night and said, "Hey. Okay, we'll take you."

Emmett Shear, my partner, had followed Paul for a while online. I didn't know anything, so I just kind of figured, "Get that money." We took it and we moved to Boston. The first batch of Y Combinator startups had eight companies, including Kiko (of course, the best and first); Loopt, started by Sam Altman; and Reddit, which was started by my friends Steve Huffman and Alexis Ohanion. Reddit is the homepage for the web and probably the most interesting and prolific web community today.



There was also Memamp, which was started by a couple of other friends of mine; ClickFacts, a malware solution company; and Infogami, a blogging software started by Aaron Swartz. We spent a year and a couple months working at Kiko and we realized we weren't very good at making a calendar, probably because we didn't use calendars. We were college students and what did we need to schedule? I only had class two days a week, so I could remember that. All we did was sit around and program, so we didn't have any appointments. We really didn't know what we were doing, and we didn't know who we were building this for.

Then, Google Calendar came out. A lot of people liked that, and we decided Kiko wasn't working. Eventually, about 14 months later, we sold it to Tucows in Toronto, Canada. We had put it on eBay and said, "Hey, maybe we can get $50,000 to pay back our investors." When I woke up on the last day of our eBay auction, there was a bid for $80,000. That's awesome. We had made no money, and the bid just kept going up every time I would refresh. It was $80,000, $113,000, $150,000, and then finally, an hour later, it was $258,000. We were pretty ecstatic. I was sitting there in my friend's apartment in my underwear refreshing an eBay page and screaming. It was pretty awesome.

One day, we were with Paul and Robert Morris, who's another partner at Y Combinator and also a famous computer science professor at MIT. Emmett and I told them we had an idea for something called JustinTV. We explained it as a crazy, camera-on-head thing where we would run around and film our own reality TV show. I remember Robert said, "I would fund that just to see you make a fool of yourself."

So, we walked out of there with a check for $50,000, and that was it. We were doing it. JustinTV was called JustinTV because I was the only one that volunteered to wear the camera.

It started off as us trying to make our own live video reality show on the web. When we launched the show, it immediately became this epicenter for pranks. One time, it was pretty serious. They had called in a stabbing in our apartment, and the cops came and kicked in the doors with guns drawn, expecting to see a stabbing victim in our apartment, but it was just us sitting there on our laptops. It was a pretty awkward situation.

We didn't know anything about creating content; most of the content we created was us sitting around on our couch using our laptops. People would text or email me, saying, "Get off your computer and go do something. Entertain us."

Pretty quickly, we realized we weren't that interesting, but we needed to do something else. We turned JustinTV into a platform for anyone to create live video content. After that, people who were much more interesting than us started broadcasting and that's when it really took off.

After JustinTV, we started a couple projects that have kind of become bigger than originally planned. The first one was Twitch, which is our gaming site — like ESPN for gaming. We now have about 20 million unique users that watch gaming content every month. That's everyone from professional Starcraft players, to people playing new releases, to gaming journalists demoing new games and doing reviews, to people just having fun.

We also spun off another company called SocialCam, which is like Instagram for videos. It's the easiest way to get video off your iPhone and share it with your friends. That's been running as an independent company outside of JustinTV. SocialCam just uploads your video and shrinks it in size, but it also transposes it to different qualities so you can watch it on different devices. The goal is to get a video to whomever you want to as fast as possible.

> **"We really didn't know what we were doing, and we didn't know who we were building this for."**

Recently, SocialCam was acquired by Autodesk for $60 million. That happened on Tuesday actually, the day after my birthday. It's been the longest road for the team, and we had 100 million Facebook users.

Exec is a company I started with my brother and a longtime friend of ours. The inspiration for Exec came from a trip to Burning Man. One of my friends had forgotten his ticket at his apartment building, which was in downtown San Francisco, so how could he get the ticket without going back? It turned out another friend of ours was driving up but was leaving in half an hour, so we needed to get that friend the key. I said, "Hey, call Uber and tell the driver to go to point A, wait for a girl and pick up a key, and then drive to point B and drop it off with the doorman." It worked. After that, I thought it would be cool to have a service like that — a service to get something done while you're busy or remote in the real world. Exec is the easiest and fastest way to get anything you want done right now. When you submit your job on an iPhone or the web, you just write a short description and then press a button. We go out and find someone for you right then. You don't have to choose the person, and there's no negotiation; it's just a flat rate of $25 an hour.

The social aspect of Exec — the way that we create jobs for people who can't find jobs right now — has been really impactful because it's something we really didn't expect. It has been really meaningful to me to have people tell us, "I would be in a really dark place right now if I didn't have this job. Thank you." That makes me feel like I'm doing something that's bigger than any project I've worked on before.

I think Exec can change the world because everyone can either be working on Exec or hiring people through Exec. We have all sorts of jobs that people can pick up and make some extra money. The future of America requires us to figure out better education and specifically job retraining, but hopefully we can do our part at Exec to help provide people with ways to make money in the interim while we're figuring that out. ■

Justin Kan is the founder and CEO of Exec, your on demand work force. Previously he founded Justin.tv, TwitchTV and Socialcam. He is a part time partner at Y Combinator.

Based on Justin Kan's interview by Interloper Films.
Reprinted with permission. First appeared in *hn.my/jkan* (justinkan.com)

Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

What Happens to Stolen Bicycles?

By ROHIN DHAR

Photo: *flickr.com/photos/narciss/3543547214/*

At Priceonomics, we are fascinated by stolen bicycles. Put simply, why the heck do so many bicycles get stolen? It seems like a crime with very limited financial upside for the thief, and yet bicycle theft is rampant in cities like San Francisco (where we are based). What is the economic incentive for bike thieves that underpins the pervasiveness of bike theft? Is this actually an efficient way for criminals to make money?

It seems as if stealing bikes shouldn't be a lucrative form of criminal activity. Used bikes aren't particularly liquid or in demand compared to other things one could steal (phones, electronics, drugs). And yet, bikes continue to get stolen, so they must be generating sufficient income for thieves. What happens to these stolen bikes, and how do they get turned into criminal income?

The Depth of the Problem

In San Francisco, if you ever leave your bike unlocked, it will be stolen. If you use a cable lock to secure your bike, it will be stolen at some point. Unless you lock your bike with medieval-esque u-locks, your bike will be stolen from the streets of most American cities. Even if you take these strong precautions, your bike may still get stolen.

According the National Bike Registry and FBI, $350 million in bicycles are stolen in the United States each year. Beyond the financial cost of the crime, it's heartbreaking to find out someone stole your bike; bikers love their bikes.

As one mom wrote in an open letter to the thief who pinched her twelve year old son's bike:

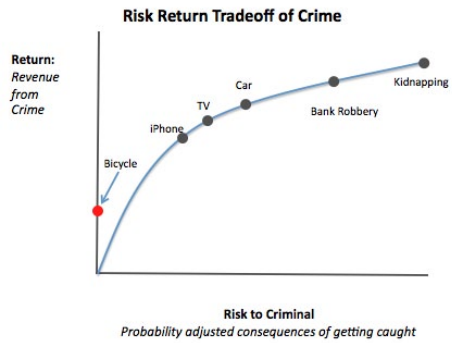
> *It took CJ three weeks to finally decide on his bike. We looked at a brown bike at Costco, even brought it home to return it the next day, and a blue one at Target. But his heart was set on the green and black Trek he saw at Libertyville Cyclery. CJ knew it was more than we wanted to spend, but the boy had never asked for anything before. You see, CJ had to live through his dad being unemployed for 18 months and knew money was tight. Besides, he's just an all around thoughtful kid.*
>
> *CJ didn't ride his bike to school if there was rain in the forecast and he always locked it up. You probably noticed that it doesn't have a scratch on it. CJ treated his bike really well and always used the kick-stand.*
>
> *You should know that CJ has cried about the bike and is still very sad. He had to learn a life lesson a little earlier than I had liked: that there are some people in the world who are just plain mean. Now you know a little about my really awesome son and the story behind his green and black Trek 3500, 16-inch mountain bike.*

An Economic Theory of Bike Crime

In 1968, Chicago economist Gary Becker introduced the notion that criminal behavior could be modeled using conventional economic theories. Criminals were just rational actors engaged in a careful cost-benefit analysis of whether to commit a crime. Is the potential revenue from the crime greater than the probability adjusted weight of getting caught? Or, as

the antagonist in the movie The Girl Next Door puts it, "Is the juice worth the squeeze?"

Criminal activity (especially crime with a clear economic incentive like theft) could therefore be modeled like any financial decision on a risk reward curve. If you are going to take big criminal risk, you need to expect a large financial reward. Crimes that generate more reward than the probability weighted cost of getting caught create expected value for the criminal. Criminals try to find "free lunches" where they can generate revenue with little risk. The government should respond by increasing the penalty for that activity so that the market equilibrates and there is an "optimal" amount of crime.



Using this risk-return framework for crime, it begins to be clear why there is so much bike theft. For all practical purposes, stealing a bike is risk-free crime. It turns out there is a near zero chance you will be caught stealing a bike and if you are, the consequences are minimal.

There are a few great accounts of journalists getting their bikes stolen and then going on a zealous mission to try to capture bikes thieves. In each account, they ultimately learn from local police that the penalty for stealing a bike is generally nothing.

"We make it easy for them. The DA doesn't do tough prosecutions. All the thieves we've busted have got probation. They treat it like a petty crime."

"You can't take six people off a murder to investigate a bike theft."

Bike thievery is essentially a risk-free crime. If you were a criminal, that might just strike your fancy. If Goldman Sachs didn't have more profitable market inefficiencies to exploit, they might be out there arbitraging stolen bikes.

What Happens to the Stolen Bikes?

Just because the risk of a crime is zero, that doesn't mean that a criminal will engage in that crime. If that were the case, thieves would go about stealing dandelions and day-old newspapers. There has to be customer demand and a liquid market for the product in order for the criminal to turn their contraband into revenue. So, how exactly does a criminal go about converting a stolen bicycle to cash?

We decided to survey the prior literature on where stolen bikes are sold as well as consult with bike shops and experts in San Francisco to get a better picture of who steals bikes and where the stolen bikes end up.

Amateur Bike Thieves. Amateur bike thieves sell their stolen goods at local fencing spots and are typically drug addicts or down on their luck homeless.

Sgt. Joe McKolsky, bike theft specialist for the SFPD, estimates that the overwhelming majority of bike thefts are driven by drug addicts and end up being sold on the street for 5 to 10 cents on the dollar. Any bike will do, whether it's a $50 beater or a $2,000 road bike. These thieves are amateurs just opportunistically stealing unsecured bikes to get some quick cash:

"Bikes are one of the four commodities of the street — cash, drugs, sex, and bikes….You can virtually exchange one for another."

In San Francisco, these amateur stolen bikes end up on the streets at the intersection of 7th Street and Market Street in front of the Carl's Jr restaurant. We chatted with Brian Smith, co-owner of HuckleBerry Bicycles, which is located across the street from this fencing joint. He confirmed it's not uncommon for people to come into the shop having just purchased a $50 bike across the street or with obviously stolen bikes.

Professional Thieves. On the other end of the spectrum are professional bike thieves. Instead of opportunistically targeting poorly locked bicycles, these thieves target expensive bicycles. They have the tools that can cut through u-locks and aim to resell stolen bikes at a price near their "fair market value." These thieves acquire the bicycles from the streets, but then resell them on online markets to maximize the selling price.

We asked Aubrey Hoermann, owner of used bicycle shop Refried Cycles in the Mission, about professional bike thieves and where they sell their merchandise:

> *"It has to end up somewhere where you can sell it in another city. My feeling is that people steal enough bikes to make it worth to take a trip somewhere like LA and then sell it there on Craigslist. If you have about 10 stolen bikes, it's probably worth the trip."*

Another bike shop proprietor who asked not to be named added:

> *"Most of these guys are drug addicts, but a lot of them are professionals. You can cut through a u-lock in a minute and a half with the right tools. Steal three bikes and sell them in LA for $1500 a piece and you're making money."*

These thieves essentially are maximizing their revenue per van trip to a market in which they can sell the bicycle. In the past they might've been able to resell it locally, but according to Aubrey, this opportunity is fading:

> *You can't just steal a bike and sell it on Craigslist in San Francisco anymore. It's too well known that's where it would be and it's too much work to change it to make it look different. I used to be a bike messenger and if your bike was stolen you'd go check at 7th and Market. Now that's too well known to just sell a bike there.*

Increasingly when a bicycle is stolen, the victims know where to check locally (Craigslist, 7th and Market, the Oakland Flea Market) so that makes it hard to sell the bikes there. Because bikes aren't even that popular in the first place, it's just not worth the effort to customize and disguise them for local sale.

Because of this dynamic, Aubrey concludes that professional bike theft is replacing amateur theft as the predominant form of bike theft. While the police may not penalize bicycle thieves, it's becoming easier for the person whose bike was stolen to investigate the bike theft themselves. This is making it harder for the amateur thief to casually flip a stolen bike.

Is There at Keyser Söze of the Bike Underworld?

Bike theft is rampant and increasingly the province of professionals. Is there any evidence that a "criminal mastermind" exists behind this network where bikes are stolen in one city, transported to another and then resold? Ultimately, there is no evidence that a bike kingpin exists.

The largest bike theft arrests ever recorded are rather mundane actually. In San Francisco, recently a local teen was arrested with hundreds of stolen bikes found in his storage locker. Did these bicycles end up in some exotic fencing ring? Nope, they were being resold at an Oakland flea market.

In Toronto, a mentally imbalanced bike shop owner was found hoarding 2,700 stolen bikes. Mostly, he was just letting them rust.

Criminal masterminds have to value their time and resources, and bike theft isn't really that profitable. The transportation costs and low value density ratio of the product likely kill the economics of the stolen bike trade. The bike shop proprietor we interviewed that requested anonymity concluded:

> *You'd be in the prostitution or drugs business if you were running a criminal ring to make money. There just isn't that much money in bikes. These people who steal bikes are professionals but small time operators. Or, they're just assholes.*

Conclusion

Ultimately, that's the point everyone seems to agree on: bike thieves are assholes. For everything else, there is little consensus and hard evidence. However, some things are clear and explain a lot of the bike theft that occurs.

It's dead simple to steal a bike and the consequences are near zero. You can resell stolen bikes. If you want to get a good price for a stolen bicycle, it requires a decent amount of work. That amount of work is what limits the bike theft trade from really flourishing. Criminal masterminds have an opportunity cost for their time; they can't be messing around lugging heavy pieces of metal and rubber that are only in limited demand.

So, if your bike ever gets stolen, you can at least take solace in the fact that the illicit bike trade isn't a very easy way to make a lot of money. That probably won't make you feel any better though. ■

Rohin Dhar is the co-founder of Priceonomics Price Guides. He is also the co-founder of Personforce job boards and has an MBA from Stanford and BA from Dartmouth. You can follow him on Twitter here @rohindhar

Reprinted with permission of the original author.
First appeared in *hn.my/stolenbikes* (priceonomics.com)

MEET MANDRILL

By MailChimp

Mandrill is a new way to send transactional, triggered, and personalized emails.
It's also the world's largest species of monkey.

MANDRILL.COM

What A Hacker Learns After A Year In Marketing

By ROB SPECTRE

A YEAR AGO LAST Friday I left eight years cutting code and plumbing servers to take my very first marketing job. Prior to then and even before in college and high school, hard skills were what paid my bills — technical work building stuff mostly for the Internet. Everything I had done up until last year required only the soft skills needed to send a group email or interview a candidate, certainly a pittance to those required to craft a message and get it in front of an audience.

I knew I needed more than that. While I was at Boxee working for Avner Ronen I made the determination that I wanted the CEO role for my startup. Like a lot of folks who spend their career in the high risk, high reward, high laughs world of early stage tech, I've long held my own entrepreneurial ambitions, but after working for a programmer-turned-head-honcho, I came around to the notion I could make a greater contribution to that endeavor by pushing the vision and the culture rather than the technology and architecture. I didn't want to be the technical co-founder; I wanted to run the circus.

But, I was sorely deficient. Sales and marketing were skills I just didn't have and were I to ask others to entrust their livelihoods and their families in such an enterprise, it would be incumbent upon me to learn. To do such a thing with a knowledge base very nearly zero would just be irresponsible.

So, to get some of those skills while keeping my technical chops up, I hopped onboard Twilio as a developer evangelist. Like a lot of companies, Twilio's devangelism program is under the marketing aegis, and the gig meant working for one of the best marketers I knew [distributionhacks.com]. I'd still write code, but would do so surrounded by the thoroughly unfamiliar context of message craft and story telling. And through the daily demands of the job and the proximity of those who do it well, hopefully I'd learn a thing or two about this marketing thing and ultimately serve those I wish to lead better.

Holy biscuits, did I learn plenty! A year in, I thought it might be helpful to my fellow developers to share what it's like to turn to the Dark Side and what I picked up in the process.

1 This Shit Is Hard

Like many folks who build stuff, my disdain for marketing as a business discipline had grown ironclad. I thought soft skills meant it was a soft job: 9 to 5 without pagers ringing, apocalyptic deadlines, or material consequences for poor workmanship. A marketer was never around when I had to get a server back up or the prod db was borked; this gig must be easy.

I learned swiftly that this view was as legitimate as assuming web development is easy after installing a Squarespace theme. My view (and likely yours) was informed mostly by bad marketing, which is every bit as prevalent as bad programming. Install ten WordPress plugins and base a view on software engineering and I'm sure the 7 out of 10 bad experiences one would

encounter would foster a belief that the entire discipline is bankrupt.

As it turns out, the ones who do it well are rare and far less visible because, like good programmers, their work is a lot harder to notice. Good marketing is a product of the same inputs as good code; long hours, sweating the details, and the judicious application of experience doing it the right way.

2 Data Wins Arguments

When debating the performance of a chunk of code or a particular architectural decision, I'd often find myself at loggerheads with my colleagues with none in the argument operating with any real evidence. And, invariably, to win I'd just test the hypothesis on a small scale, show the comparative data, and the decision would be much clearer.

Sometimes I was right, sometimes I was wrong. But the practice of testing intuition on reduced scope to gain confidence about a decision is one I use every day as an evangelist. And, as it turns out, it is a practice used by every person good at marketing. "It's all a numbers game," people would tell me, leading me to believe that I'd be spending a lot of time in spreadsheets fiddling with a formula until it did what I wanted. Surely those charts and graphs meant nothing, and at the end of the day a small amount of math could be twisted to support my own preconceptions. "Developers don't want a bunch of examples," I'd say. "Just give them really strong reference documentation, and they'll figure it out."

Not so. Marketing data shows in stark relief what works and what doesn't and — especially when working on the Internet — is readily available if you spend a little effort trying to find it. Folks with a technical background excel at such, and wielding that power in this discipline can yield very powerful results, if less powerful buzzwords.

3 Calendar Management Is A Skill

Managing my meetings was by far the most difficult part of my first few months as a developer evangelist. When I was writing code, meetings were always something I could punt. When a reminder would come in and I didn't feel like being bothered, I could always throw some headphones on, spit out a quick email about needing to stay heads down on a problem, and everyone would just magically wait until I was ready for them. People came to me.

Man, those were the days. A lot of marketing is gently aligning external forces to craft the right message and get it in front of the right people at the right time. And since those external forces don't need me for a login page or a bug fix, they are far less inclined to tolerate last minute pushes or tardiness.

I must have run up and down Manhattan every day the first month I was at Twilio. I'd set a meeting at 42nd and Broadway next to one at Fulton and Church with 15 minutes in between. I'd double and triple book in email, leaving two or three of the parties asking where the hell I was. This function that had always been a nuisance in my life was now a critical skill, and I found out I sucked at it.

Took a long while to learn. I'm still not very good at it.

4 You Can Learn To Schmooze

I'm not naturally very charismatic or talkative. Despite having played in a band and given a fair number of technical presentations, it's just not something I have a genetic talent for, and I have to work very hard to do it. But in evangelism, this is part and parcel of the profession and indeed a valuable ability in the marketing game.

And, much to my delight, it is something you do get better at with practice. Programming is something I felt I could always just do. But public speaking, networking at a party, meeting people at a conference just never came as easily to me as writing code. It is now something I feel I can do and do well, and the only difference was a lot of practice.

There aren't any real secrets. Ask people what they are working on, always treat them not as a means but an end, and be your authentic, flawed, fully present self. Nearly every human you meet will respond kindly. And those who don't, you just don't have to worry about.

It's hard, but so is learning Erlang. And just like you cringe when you revisit the first Post-Nuke you ever built, so too will you when you recall your first attempts at building your interpersonal skills (just ask the kids at PennApps about my first Twilio demo. What a bomb on stilts that was).

Don't get discouraged. Just grit your teeth, plow through and practice. You will get better.

5 The Impact You Can Make Is Huge

I long thought my maximum point of power to effect real change was in the text editor in front of me. The only way I could make an impact on people's lives at scale was to write great software. While I still think we who can write code wield awesome power indeed, I've learned more parts of a startup than just engineering can make a huge impact.

While in the thick of the Olympics of hustling called SXSW, my paths crossed into a coder from LA named Will. He gave me a high-five for my Twilio shirt and said he was working on an app that would let people create disposable phone numbers to use for Craiglist posts, job interviews, and other calls you needed screened. My somewhat flippant question after hearing about his product was, "When you going to ship?"

"Soon, soon," he said. "We're working on it."

"Well, hurry up!" I exclaimed. "People need this!"

A few months later he and his crew at AdHoc did ship that app, launched it on HackerNews, and the response has been incredible. So incredible in fact only a few days after launch, it helped a guy in Portland catch the thief who stole his bike.

After the launch, I got a very kind thank you from Will for the little push to get his app shipped. The right message at the right time to the right person helped encourage an intrepid team to finish a great idea, earn a lot of business, and help a dude I'll never meet got his bike back.

Now I can code all goddamn day and probably never achieve the same impact as that little conversation in the middle of a busy conference. Just a little encouragement at the right moment helped a team build something of which they are rightfully proud and serve some people who needed it. The satisfaction I got from watching that squad's product blow up on the news was immense.

And when I'm doing this marketing thing right, that's what it always feels like. I was anticipating a lot of different outcomes starting down this path, but I didn't expect it to feel so rewarding. Good marketing is tough to do, good programming is tough to do; I'm starting to learn that anything good is tough to do.

And, for this hacker at least, doing something well will always feel magical. ■

Doing just about anything for a good laugh, Rob runs developer evangelism for Twilio and is an ardent supporter of open source software and creative commons art, the startup scene in New York, and every professional sports club from Boston. In addition to writing on Brookyn Hacker, he runs a number of exploits into Internet ridiculousness, including the heartwarming documentary service how i knew you were the one, the robotic telephonic joke machine Laugh-o-Tron, and the Nobel Prize-losing Chrome Extension Jeter Filter.

Reprinted with permission of the original author.
First appeared in *hn.my/hacketer* (brooklynhacker.com)

Push, Push, Push

Expanding Your Comfort Zone

By DEREK SIVERS

I'm 40 meters underwater. It's getting cold and dark. It's only the third dive in my life, but I'm taking the advanced training course, and the Caribbean teacher was a little reckless, dashing ahead, leaving me alone.

The next day I'm in a government office, answering an interview, raising my right hand, becoming a citizen of Dominica.

I'm in a Muslim Indian family's house in Staten Island, washing my feet, with the Imam waiting for my conversion ceremony. Next week they will be my family in-law. The Muslim wedding will make her extended family happy. I've memorized the syllables I need to say. "Ash hadu alla ilaha illallah. Ash hadu anna muhammadar rasulullah."

I'm backstage at the TED Conference, about to go on, but I can't remember my lines. In the audience are Bill Gates, Al Gore, Peter Gabriel, and a few hundred other intimidating geniuses. Heart pounding so fast and hard, I think I'm going to explode. They call my name. Ack! I still can't remember my lines! But I hit the stage anyway.

I'm alone on a bicycle in a forest in Sweden. I left from Stockholm 6 hours ago, headed south, with only 50 kronor, and I'm getting hungry. I don't know the way back.

We're in a filthy dorm-room apartment in Guilin, China, studying at the local university. At the local grocery store, we choose from a bin of live frogs.

The India Embassy official hands me a pseudo-passport that says I am now officially a "Person of Indian Origin," a pseudo-citizen of India.

I'm in the back of a truck in Cambodia, soaking wet, hitching a ride back to Phnom Penh after an all-day bike ride. The roads were flooded, but we rode our bikes through anyway, Mekong River water waist-high.

That week I speak at four conferences in Cambodia, Singapore, Brunei, and Indonesia. By the 4th one, my American accent has started to morph into something kind of Asian.

We're in a hospital in Singapore, having a baby. It's a boy, which means he'll serve 2 years in the Singapore military in 2030. The birth certificate says his race is Eurasian, a word I've never heard.

I'm on a diplomatic mission in Mongolia, with the Singapore Business Federation, talking with the Mongolian government's head of business development, walking with the next mayor of Ulaanbaatar.

I suppress a laugh at the ridiculousness of this situation.

I'm just a musician from California! What the hell am I doing here?

But that feeling lets me know I'm on the right track. This is exactly what I wanted.

Some people push themselves physically, to see how far they can go. I've been doing the same thing culturally, trying to expand my California-boy perspective.

I love that when we push push push, we expand our comfort zone. Things that used to feel intimidating now are as comfortable as home.

I remember how scary New York City felt when I moved there in 1990, just 20 years old. Two years later it was "my" city, my comfort zone.

Now previously-exotic Singapore is my long-term comfortable home, while I push myself into exploring foreign places, new businesses, and different perspectives.

After years of stage fright, performing over 1000 shows, I have a strong case of "stage comfort." Being the lead singer or speaker on stage is now my comfort zone.

A lot of my musician friends feel this when playing on stage with their legendary heroes. You push push push, and then one day find yourself on the very stage you used to dream about. And it feels so natural — almost relaxing. It's your new comfort zone.

The question is: what scares you now? What's intimidating? What's the great unknown?

I keep using that question to guide my next move. ■

Derek Sivers founded a music distribution company, CD Baby, in 1997, a web hosting company, Hostbaby, in 2000, and sold both in 2008. Since then he's been a popular speaker at the TED Conferences, and writing short essays at *sivers.org*

Reprinted with permission of the original author.
First appeared in *hn.my/push* (sivers.org)

Using Python and *k*-means to Find the Dominant Colors in Images

By CHARLES LEIFER

I'm working on a little photography website for my Dad and thought it would be neat to extract color information from photographs. I tried a couple of different approaches before finding one that works pretty well. This approach uses k-means clustering [hn.my/kmeans] to cluster the pixels in groups based on color. The center of those resulting clusters is then the "dominant" color(s). k-means is a great fit for this problem because it is (usually) fast, but the caveat is that it requires you to specify up-front how many clusters you want — I found that it works well when I specified around 3.

A warning

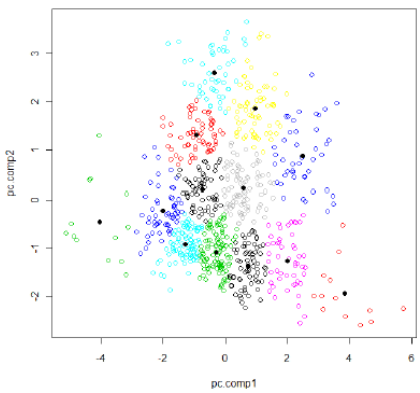
I'm no expert on data-mining — almost all my experience comes from reading Toby Segaran's excellent book Programming Collective Intelligence. In one of the first chapters Toby covers clustering algorithms, including a nice treatment of k-means, so if you want to really learn from an expert I'd suggest picking up a copy. You won't be disappointed.

How it works

The way I understand it to work is you start with a bunch of data points. For simplicity, let's say they're numbers on a number-line. You want to group the numbers into "k" clusters, so pick "k" points randomly from the data to use as your "clusters."

Now, loop over every point in the data and calculate its distance to each of the "k" clusters. Find the nearest cluster and associate that point with the cluster. When you've looped over all the points they should all be assigned to one of the "k" clusters. Now, recalculate each cluster's center by averaging the distances of all the associated points and start over.

When the centers stop moving very much you can stop looping. You will end up with something like this: the points are colored based on what "cluster" they are in and the dark-black circles indicate the centers of each cluster.



Applying it to photographs

The neat thing about this algorithm is that, since it relies only on a simple distance calculation, you can extend it out to multi-dimensional data. Color is often represented using 3 channels: Red, Green, and Blue. So what I did was treat all the pixels in the image like points on a 3-dimensional space. That's all there was to it!

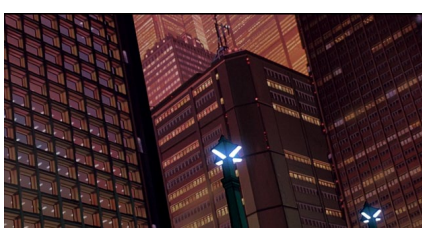
I made a few optimizations along the way:

1. Resize the image down to 200 x 200 or so using PIL [hn.my/pil]
2. Instead of storing "duplicate" points, store a count with each — saves on calculations

Looking at some results

The result: 

The result: 



The result: 

The source code

Below is the source code. It requires PIL to resize the image down to 200x200 and to extract the colors/counts. The `colorz` function is the one that returns the actual color codes for a filename.

```
from collections import namedtuple
from math import sqrt
import random
try:
    import Image
except ImportError:
    from PIL import Image

Point = namedtuple('Point', ('coords', 'n', 'ct'))
Cluster = namedtuple('Cluster', ('points', 'center', 'n'))

def get_points(img):
    points = []
    w, h = img.size
    for count, color in img.getcolors(w * h):
        points.append(Point(color, 3, count))
    return points

rtoh = lambda rgb: '#%s' % ''.join(('%02x' % p for p in rgb))

def colorz(filename, n=3):
    img = Image.open(filename)
    img.thumbnail((200, 200))
    w, h = img.size

    points = get_points(img)
    clusters = kmeans(points, n, 1)
    rgbs = [map(int, c.center.coords) for c in clusters]
    return map(rtoh, rgbs)

def euclidean(p1, p2):
    return sqrt(sum([
        (p1.coords[i] - p2.coords[i]) ** 2 for i in range(p1.n)
    ]))

def calculate_center(points, n):
    vals = [0.0 for i in range(n)]
    plen = 0
    for p in points:
        plen += p.ct
        for i in range(n):
            vals[i] += (p.coords[i] * p.ct)
    return Point([(v / plen) for v in vals], n, 1)
```

```
def kmeans(points, k, min_diff):
    clusters = [Cluster([p], p, p.n) for p in random.sample(points, k)]

    while 1:
        plists = [[] for i in range(k)]

        for p in points:
            smallest_distance = float('Inf')
            for i in range(k):
                distance = euclidean(p, clusters[i].center)
                if distance < smallest_distance:
                    smallest_distance = distance
                    idx = i
            plists[idx].append(p)

        diff = 0
        for i in range(k):
            old = clusters[i]
            center = calculate_center(plists[i], old.n)
            new = Cluster(plists[i], center, old.n)
            clusters[i] = new
            diff = max(diff, euclidean(old.center, new.center))

        if diff < min_diff:
            break

    return clusters
```

Playing with it in the browser

I ported the code over to JavaScript — let me tell you, it's pretty rough, but it works and is fast. If you'd like to take a look at a live example, check out:

charlesleifer.com/static/colors/

You can view the source to see the JavaScript version, but basically it is just using the HTML5 canvas and its `getImageData` method. ■

Charles Leifer is a Python developer both professionally and for his own open source projects [github.com/coleifer]. He previously worked three and a half years for Mediaphormedia, the company responsible for creating the Django framework.

Reprinted with permission of the original author.
First appeared in *hn.my/color* (charlesleifer.com)

"The Best Programming Advice I Ever Got"

By ROB PIKE

A YEAR OR TWO after I'd joined the Labs, I was pair programming with Ken Thompson on an on-the-fly compiler for a little interactive graphics language designed by Gerard Holzmann. I was the faster typist, so I was at the keyboard and Ken was standing behind me as we programmed. We were working fast, and things broke, often visibly — it was a graphics language, after all. When something went wrong, I'd reflexively start to dig in to the problem, examining stack traces, sticking in print statements, invoking a debugger, and so on. But Ken would just stand and think, ignoring me and the code we'd just written. After a while I noticed a pattern: Ken would often understand the problem before I would, and would suddenly announce, "I know what's wrong." He was usually correct. I realized that Ken was building a mental model of the code and when something broke it was an error in the model. By thinking about *how* that problem could happen, he'd intuit where the model was wrong or where our code must not be satisfying the model.

Ken taught me that thinking before debugging is extremely important. If you dive into the bug, you tend to fix the local issue in the code, but if you think about the bug first, how the bug came to be, you often find and correct a higher-level problem in the code that will improve the design and prevent further bugs.

I recognize this is largely a matter of style. Some people insist on line-by-line tool-driven debugging for everything. But I now believe that thinking — without looking at the code — is the best debugging tool of all, because it leads to better software. ■

Rob Pike is a Distinguished Engineer at Google, Inc. He works on distributed systems, data mining, programming languages, and software development tools. Most recently he has been a co-designer and developer of the Go programming language.

Reprinted with permission of the original author.
First appeared in *hn.my/bestadvice* (informit.com)

Effective Text Editing

By BRAM MOOLENAAR

If you spend a lot of time typing plain text, writing programs or writing HTML, you can save much of that time by using a good editor and using it effectively. This article will present guidelines and hints for doing your work more quickly and with fewer mistakes.

The open source text editor Vim (Vi IMproved) will be used here to present the ideas about effective editing, but they apply to other editors just as well. Choosing the right editor is actually the first step towards effective editing. The discussion about which editor is the best for you would take too much room and is avoided. If you don't know which editor to use or are dissatisfied with what you are currently using, give Vim a try; you won't be disappointed.

PART 1: EDIT A FILE

1 Move around quickly

Most time is spent reading, checking for errors and looking for the right place to work on, rather than inserting new text or changing it. Navigating through the text is done very often, thus you should learn how to do that quickly.

Quite often you will want to search for some text you know is there. Or look at all lines where a certain word or phrase is used. You could simply use the search command `/pattern` to find the text, but there are smarter ways:

- If you see a specific word and want to search for other occurrences of the same word, use the `*` command. It will grab the word from under the cursor and search for the next one.
- If you set the `'incsearch'` option, Vim will show the first match for the pattern, while you are still typing it. This quickly shows a typo in the pattern.
- If you set the `'hlsearch'` option, Vim will highlight all matches for the pattern with a yellow background. This gives a quick overview of where the search command will take you. In program code, it can show where a variable is used. You don't even have to move the cursor to see the matches.

In structured text there are even more possibilities to move around quickly. Vim has specific commands for programs in C (and similar languages like C++ and Java):

- Use `%` to jump from an open brace to its matching closing brace. Or from a "#if" to the matching "#endif." Actually, `%` can jump to many different matching items. It is very useful to check `if ()` and `{}` constructs are balanced properly.
- Use `[{` to jump back to the "{" at the start of the current code block.
- Use `gd` to jump from the use of a variable to its local declaration.

There are many more, of course. The point is that you need to get to know these commands. You might object that you can't possibly learn all these commands — there are hundreds of different movement commands, some simple, some very clever — and it would take weeks of training to learn them all. Well, you don't need to. Instead, realize what your specific way of editing is, and learn only those commands that make your editing more effective.

There are **three basic steps**:

1. While you are editing, keep an eye out for actions you repeat and/or spend quite a bit of time on.
2. Find out if there is an editor command that will do this action quicker. Read the documentation, ask a friend, or look at how others do this.
3. Train using the command. Do this until your fingers type it without thinking.

Let's use an example to show how it works:

1. You find that when you are editing C program files, you often spend time looking for where a function is defined. You currently use the `*` command to search for other places where the function name appears, but you end up going through a lot of matches for where the function is used instead of defined. You get the idea that there must be a way to do this faster.
2. Looking through the quick reference you find a remark about jumping to tags. The documentation shows how this can be used to jump to a function definition. Just what you were looking for!
3. You experiment a bit with generating a tags file, using the ctags program that comes with Vim. You learn to use the `CTRL-]` command and find you save lots of time using it. To make it easier, you add a few lines to your Makefile to automatically generate the tags file.

A couple of things to watch out for when you are using these three steps:

- "I want to get the work done. I don't have time to look through the documentation to find some new command." If you think like this, you will get stuck in the stone age of computing. Some people use Notepad for everything, and then wonder why other people get their work done in half the time.
- Don't overdo it. If you always try to find the perfect command for every little thing you do, your mind will have no time left to think about the work you were actually doing. Just pick out those actions that take more time than necessary, and train the commands until you don't need to think about it when using them. Then you can concentrate on the text.

In the following sections there will be suggestions for actions that most people have to deal with. You can use these as inspiration for using the **three basic steps** for your own work.

Don't type it twice

There is a limited set of words we type, and even a limited number of phrases and sentences, especially in computer programs. Obviously, you don't want to type the same thing twice.

Very often you will want to change one word into another. If you need to do this for the whole file, you can use the `:s` (substitute) command. If only a few locations need changing, a quick method is to use the `*` command to find the next occurrence of the word and use `cw` to change the word. Then, type `n` to find the next word and `.` (dot) to repeat the `cw` command.

The `.` command repeats the last change. A change, in this context, is inserting, deleting or replacing text. Being able to repeat this is a very powerful mechanism. If you organize your editing around it, many changes will become a matter of hitting just that `.` key. Watch out for making other changes in between because it will replace the change that you were repeating. Instead, you might want to mark the location with the m command, continue your repeated change and come back there later.

Some function and variable names can be awkward to type. Can you quickly type "XpmCreatePixmapFromData" without a typo and without looking it up? Vim has a completion mechanism that makes this a whole lot easier. It looks up words in the file you are editing, and also in #include'd files. You can type "XpmCr" and then hit `CTRL-N`, and Vim will expand it to "XpmCreatePixmapFromData" for you. Not only does this save quite a bit of typing, it also avoids making a typo and having to fix it later when the compiler gives you an error message.

When you are typing a phrase or sentence multiple times, there is an even quicker approach. Vim has a mechanism to record a macro. You type `qa` to start recording into register "a." Then you type your commands as usual and finally hit `q` again to stop recording. When you want to repeat the recorded commands, you type `@a`. There are 26 registers available for this.

With recording you can repeat many different actions, not just inserting text. Keep this in mind when you know you are going to repeat something.

One thing to watch out for when recording, however, is that the commands will be played back exactly as you typed them. When moving around you must keep in mind that the text you move over might be different when the command is repeated. Moving four characters left might work for the text where you are recording, but it might need to be five characters where you repeat the commands. It's often necessary to use commands to move over text objects (words, sentences) or move to a specific character.

When the commands you need to repeat are getting more complicated, typing them right at once is getting more difficult. Instead of recording them, you should then write a script or macro. This is very useful to make templates for parts of your code; for example, a function header. You can make this as clever as you like.

3 Fix it when it's wrong

It's normal to make errors while typing — nobody can avoid it. The trick is to quickly spot and correct them. The editor should be able to help you with this, but you need to tell it what's wrong and what's right.

Very often you will make the same mistake again and again because your fingers just don't do what you intended. This can be corrected with abbreviations. A few examples:

```
:abbr Lunix Linux
:abbr accross across
:abbr hte the
```

The words will be automatically corrected just after you type them.

The same mechanism can be used to type a long word with just a few characters. This is especially useful for words that you find hard to type, and it avoids that you type them wrong. Examples:

```
:abbr pn penguin
:abbr MS Mandrake Software
```

However, these tend to expand to the full word when you don't want it, which makes it difficult when you really want to insert "MS" in your text. It is best to use short words that don't have a meaning of their own.

To find errors in your text, Vim has a clever highlighting mechanism. This was actually meant to be used for syntax highlighting of programs, but it can catch and highlight errors as well.

Syntax highlighting shows comments in color. That doesn't sound like an important feature, but once you start using it you will find that it helps a lot. You can quickly spot text that should be a comment but isn't highlighted as such (you probably forgot a comment marker), or see a line of code highlighted as comment (you forgot to insert a "*/"). These are errors which are hard to spot in a B&W file and can waste a lot of time when trying to debug the code.

The syntax highlighting can also catch unbalanced braces. An unbalanced ")" is highlighted with a bright red background. You can use the % command to see how they match, and insert a "(" or ")" at the right position.

Other common mistakes are also quickly spotted. For example, using "#included <stdio.h>" instead of "#include <stdio.h>". You easily miss the mistake in B&W, but quickly spot that "include" is highlighted while "included" isn't.

A more complex example: for English text there is a long list of all words that are used. Any word not in this list could be an error. With a syntax file, you can highlight all words that are not in the list. With a few extra macros, you can add words to the wordlist, so that they are no longer flagged as an error. This works just as you would expect in a word processor. In Vim it is implemented with scripts and you can further tune it for your own use; for example, to only check the comments in a program for spelling errors.

PART 2: EDIT MORE FILES

4 A file seldom comes alone

People don't work on just one file. Mostly there are many related files, and you edit several after each other, or even several at the same time. You should be able to take advantage of your editor to make working with several files more efficient.

The previously mentioned tag mechanism also works for jumping between files. The usual approach is to generate a tags file for the whole project you are working on. You can then quickly jump between all files in the project to find the definitions of functions, structures, typedefs, etc. The time you save compared with manually searching is tremendous; creating a tags file is the first thing I do when browsing a program.

Another powerful mechanism is to find all occurrences of a name in a group of files using the `:grep` command. Vim makes a list of all matches and jumps to the first one.

The `:cn` command takes you to each next match. This is very useful if you need to change the number of arguments in a function call.

Include files contain useful information, but finding the one that contains the declaration you need to see can take a lot of time. Vim knows about include files and can search them for a word you are looking for. The most common action is to lookup the prototype of a function. Position the cursor on the name of the function in your file and type `[I:`. Vim will show a list of all matches for the function name in included files. If you need to see more context, you can directly jump to the declaration. A similar command can be used to check if you did include the right header files.

In Vim you can split the text area in several parts to edit different files. Then you can compare the contents of two or more files and copy/paste text between them. There are many commands to open and close windows, jump between them, temporarily hide files, etc. Again you will have to use the three basic steps to select the set of commands you want to learn to use.

There are more uses of multiple windows. For example, the preview-tag mechanism is a good feature. This opens a special preview window while keeping the cursor in the file you are working on. The text in the preview window shows, for example, the function declaration for the function name that is under the cursor. If you move the cursor to another name and leave it there for a second, the preview window will show the definition of that name. It could also be the name of a structure or a function which is declared in an include file of your project.

5 Let's work together

An editor is for editing text. An e-mail program is for sending and receiving messages. An Operating System is for running programs. Each program has its own task and should be good at it. The power comes from having the programs work together.

A simple example: You need to write a summary of no more than 500 words. Select the current paragraph and write it to the "wc" program: `vip:w !wc -w`. The external "wc -w" command is used to count the words. Easy, isn't it?

There will always be some functionality that you need that is not in the editor. Making it possible to filter text with another program means you can add that functionality externally. It has always been the spirit of Unix to have separate programs that do their job well and work together to perform a bigger task. Unfortunately, most editors don't work too well together with other programs. You can't replace the e-mail editor in Netscape with another one, for example. You end up using a crippled editor. Another tendency is to include all kinds of functionality inside the editor; Emacs is a good example of where this can end up. (Some call it an operating system that can also be used to edit text.)

Vim tries to integrate with other programs, but this is still a struggle. Currently it's possible to use Vim as the editor in MS-Developer Studio and Sniff. Some e-mail programs that support an external editor, like Mutt, can use Vim. Integration with Sun Workshop is being worked on. Generally, this is an area that has to be improved in the near future. Only then will we get a system that's better than the sum of its parts.

6 Text is structured

You will often work with text that has some kind of structure, but different from what is supported by the available commands. Then you will have to fall back to the "building blocks" of the editor and create your own macros and scripts to work with this text. We are getting to the more complicated stuff here.

One of the simpler things is to speed up the edit-compile-fix cycle. Vim has the `:make` command, which starts your compilation, catches the errors it produces and lets you jump to the error locations to fix the problems. If you use a different compiler, the error messages will not be recognized. Instead of going back to the old "write it down" system, you should adjust the `'errorformat'` option. This tells Vim what your errors look like and how to get the file name and line number out of them. It works for the complicated gcc error messages, thus you should be able to make it work for almost any compiler.

Sometimes adjusting to a type of file is just a matter of setting a few options or writing a few macros. For example, to jump around manual pages, you can write a macro that grabs the word under the cursor, clears the buffer and then reads the manual page for that word into the buffer. That's a simple and efficient way to lookup cross-references.

Using the three basic steps, you can work more effectively with any sort of structured file. Just think about the actions you want to do with the file, find the editor commands that do it and start using them. It's really as simple as it sounds; you just have to do it.

PART 3: SHARPEN THE SAW

7 Make it a habit

Learning to drive a car takes effort. Is that a reason to keep driving your bicycle? No, you realize you need to invest time to learn a skill. Text editing isn't different. You need to learn new commands and turn them into a habit.

On the other hand, you should not try to learn every command an editor offers. That would be a complete waste of time. Most people only need to learn 10 to 20 percent of the commands for their work, but it's different for everyonr. It requires that you lean back now and then and wonder if there is some repetitive task that could be automated. If you do a task only once and don't expect having to do it again, don't try to optimize it. But you probably realize you have been repeating something several times in the last hour. Then search the documentation for a command that can do it quicker. Or write a macro to do it. When it's a larger task, like lining out a specific sort of text, you could look around in newsgroups or on the Internet to see if somebody already solved it for you.

The essential basic step is the last one. You can think of a repetitive task, find a nice solution for it and after the weekend forgot how you did it. That doesn't work. You will have to repeat the solution until your fingers do it automatically. Only then will you reach the efficiency you need. Trying to learn too many things at once won't work, but doing a few at the same time will work well. For tricks you don't use often enough to get them in your fingers, you might want to write them down to look them up later. Anyway, if you keep the goal in view, you will find ways to make your editing more and more effective.

One last remark to remind you of what happens when people ignore all the above: I still see people who spend half their day behind a VDU looking up at their screen, then down at two fingers, then up at the screen, etc., and then they wonder why they get so tired... Type with ten fingers! It's not just faster but also is much less tiresome. Using a computer program for one hour each day, it only takes a couple of weeks to learn to touch-type. ■

Bram Moolenaar is the main author of Vim. He writes the core Vim functionality and selects what code submitted by others is included. He mainly works on software, but he still knows how to handle a soldering iron. He is founder and treasurer of ICCF Holland, which helps orphans in Uganda.

Reprinted with permission of the original author.
First appeared in *hn.my/habits* (moolenaar.net)

Give a Damn

By CLAY ALLSOPP

I WANT TO TELL you about The Kid. I met The Kid a few years ago, right out of high school; he had shipped some popular iPhone apps, made a few websites, and had a bright future.

I don't know how it started, but The Kid really believed in "Move Fast and Break Things." He was a ship-first-questions-later sort of guy. It made sense to him: the product was the purpose, and the code was a means to an end. He loved the things he could build, but he wasn't big on the process.

So, The Kid carried on and built a lot of cool stuff. I saw some of the code; it wasn't pretty, but the end result still worked fine. And it got him pretty far, too. He'd flaunt these creations and eyes would go wide because it all looked impressive. He was young, sure, but who wouldn't want to grab that talent while it was cheap?

The Kid started working. He was famous for shipping new features that users loved, and damn could he do it fast! It usually took just a few days from idea to production — hundreds of lines of code in an afternoon, I kid you not. Lots of pats on his back, I'm sure. It all seemed to be working out for him, living the good life.

And then things changed. I saw The Kid just about a year ago, working feverishly on a complete product redesign. It was lots of new code and not a lot of time to think about it. Just him on the project — no second opinions or supervision. As with all redesigns, feature requests piled up at the pace of a bad game of Tetris. But who was The Kid to say no? Sleepless nights later, The Kid emerged with something. At first glance it looked great; even I couldn't believe it came together so quickly. Champion effort on his part, right?

But here's the rub: beneath the surface, it was just too buggy. And these weren't just sloppy edge-case bugs; they were "What idiot do we need to fire?" class problems. The redesign was shelved and rewritten again without The Kid.

The Kid didn't lose his job, but I could tell it hurt him like hell. Because to programmers like us, what is our work but extensions of ourselves? What did this disaster say about The Kid?

He laid low for a bit, ashamed of what he had done. Moving fast and breaking things had gotten him far, but now he had finally broken too much. Kind of world-shattering to him, I guess. It was a dark place for The Kid.

And that's when I heard The Kid grew up. You could say he became The Guy, The Dude, whatever. The point is he had a change of heart. He started to realize shipping might not be everything, and his screw-up was a loud wakeup call that he needed to change his scene.

And so (and this is all hearsay, mind you), The Kid started caring about his code. Not just caring, but really giving a damn about it, and not because it was a means to an end, but for the sole sake of caring about it. "Code is more than just a tool," I heard he said. "It's our craft. It's our muscle. And we need to train it. Chop wood. Carry water. Code."

I heard all sorts of wild rumors that The Kid started using "best practices" in all his Google searches. I even heard he started learning the deep internals of the beasts he wrangled, whether it was Rails or iOS or whatever, just for the intellectual pleasure of it all. Code was no longer a beast to be tamed; it was a creature, to be both studied and admired. He even tried to teach others the error of his old ways. Wild stuff, right?

Did The Kid completely abandon his old ways? Well, apparently not. He said something about how there's a "time and place for everything" — that sometimes we need to ship fast and break things. But if we take all the other time we have and put it to good use by really learning and crafting our code, we'll break less.

That's kind of a crazy change, but I'd believe it. I thought a lot of things about The Kid when I met him years ago, but I didn't think he was stupid. He grew and evolved as we all do, and he's probably not even done yet, wherever he is. But next time you need to move fast, take a deep breath before taking the dive and remember The Kid in all of us. ■

Clay Allsopp is a hacker, Thiel Fellow, and internet enthusiast. An iOS developer since day one, Clay has crafted beautiful mobile apps with over a million cumulative downloads for startups like Circle. He is currently building Propeller [usepropeller.com], the best way for anyone to build a mobile app.

Reprinted with permission of the original author. First appeared in *hn.my/kid* (clayallsopp.com)

How To Set Up Your Linode For Maximum Awesomeness

By FEROSS ABOUKHADIJEH

I've set up at least five new servers with Linode [linode.com] and each time I complete the ritual, I learn new incantations that make the Linux angels sing. I'm pretty happy with my current recipe.

Setting up a new server can be confusing, so using a tutorial like this one is a good idea the first time you do it.

In this guide, I will demonstrate how to set up a fresh Ubuntu server from scratch, update everything, install essential software, lock down the server to make it more resilient against basic attacks and denial-of-service, improve server stability, setup automatic backups to another server, and finally install common software like Nginx, MySQL, Python, Node, etc.

Provision a New Linode

First, you need to provision a new Linode. Using Linode's web UI, it's quite easy. Select your desired Linode size. If you're unsure, choose the smallest size. You can always resize it later. Select "Ubuntu 12.04 LTS" as your OS. You'll be asked to create a password for the `root` user.

After a few minutes, your server will be ready. Now, it's time to connect to it!

Connecting to Your Server

First, open Terminal on your Mac. On Windows, you'll want to use putty [hn.my/putty], since Windows doesn't come with a proper terminal.

To connect to your server, type this into your terminal and hit Enter:

```
ssh root@<your server ip>
```

Of course, replace `<your server ip>` with your Linode's actual IP address, which you can find on the "Remote Access" tab in the control panel.

This command launches the SSH program and asks it to connect to your server with the username `root`, which is the default Ubuntu user. You will be prompted for the `root` password you created earlier.

Basic Ubuntu Setup

To set up your new server, execute the following commands.

Set the hostname

Set the server hostname. Any name will do — just make it memorable. In this example, I chose "future".

```
echo "future" > /etc/hostname
hostname -F /etc/hostname
```

Let's verify that it was set correctly:

```
hostname
```

Set the fully-qualified domain name

Set the FQDN of the server by making sure the following text is in the /etc/hosts file:

```
127.0.0.1  localhost.localdomain   localhost
127.0.1.1  ubuntu
<your server ip> future.<domain>.net  future
```

It is useful if you add an A record that points from some domain you control (in this case I used "future.<domain>.net") to your server IP address. This way, you can easily reference the IP address of your server when you SSH into it, like so:

```
ssh future.<your domain>.net
```

Set the time

Set the server timezone:

```
dpkg-reconfigure tzdata
```

Verify that the date is correct:

```
date
```

Update the server

Check for updates and install:

```
aptitude update
aptitude upgrade
```

Basic Security Setup

Create a new user

The root user has a lot of power on your server. It has the power to read, write, and execute any file on the server. It's not advisable to use root for day-to-day server tasks. For those tasks, use a user account with normal permissions.

Add a new user:

```
adduser <your username>
```

Add the user to the sudoers group:

```
usermod -a -G sudo <your username>
```

This allows you to perform actions that require root privilege by simply prepending the word sudo to the command. You may need to type your password to confirm your intentions.

Login with new user:

```
exit
ssh <your username>@<your server ip>
```

Set up SSH keys

SSH keys allow you to login to your server without a password. For this reason, you'll want to set this up on your primary computer (definitely not a public or shared computer!). SSH keys are very convenient and don't make your server any less secure.

If you've already generated SSH keys before (maybe for your GitHub account?), then you can skip the next step.

Generate SSH keys

Generate SSH keys with the following command:

(NOTE: Be sure to run this on your local computer — not your server!)

```
ssh-keygen -t rsa -C "<your email address>"
```

When prompted, just accept the default locations for the keyfiles. Also, you'll want to choose a nice, strong password for your key. If you're on Mac, you can save the password in your keychain so you won't have to type it in repeatedly.

Now you should have two keyfiles, one public and one private, in the `~/.ssh` folder.

Copy the public key to server

Now, copy your public key to the server. This tells the server that it should allow anyone with your private key to access the server. This is why we set a password on the private key earlier.

From your local machine, run:

```
scp ~/.ssh/id_rsa.pub <your username>@
<your server ip>:
```

On your Linode, run:

```
mkdir .ssh
mv id_rsa.pub .ssh/authorized_keys
chown -R <your username>:<your username> .ssh
chmod 700 .ssh
chmod 600 .ssh/authorized_keys
```

Disable remote root login and change the SSH port

Since all Ubuntu servers have a `root` user and most servers run SSH on port 22 (the default), criminals often try to guess the `root` password using automated attacks that try many thousands of passwords in a very short time. This is a common attack that nearly all servers will face.

We can make things substantially more difficult for automated attackers by preventing the `root` user from logging in over SSH and changing our SSH port to something less obvious. This will prevent the vast majority of automatic attacks.

Disable remote root login and change SSH port:

```
sudo nano /etc/ssh/sshd_config
```

Set "Port" to "44444" and "PermitRootLogin" to "no". Save the file and restart the SSH service:

```
sudo service ssh restart
```

In this example, we changed the port to 44444. So, now to connect to the server, we need to run:

```
ssh <your username>@future.<your domain>.net -p
44444
```

Advanced Security Setup

Prevent repeated login attempts with Fail2Ban

Fail2Ban [fail2ban.org] is a security tool to prevent dictionary attacks. It works by monitoring important services (like SSH) and blocking IP addresses which appear to be malicious (i.e. they are failing too many login attempts because they are guessing passwords).

Install Fail2Ban:

```
sudo aptitude install fail2ban
```

Configure Fail2Ban:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/
jail.local
sudo nano /etc/fail2ban/jail.local
```

Set "enabled" to "true" in the [ssh-ddos] section. Also, set "port" to "44444" in the [ssh] and [ssh-ddos] sections. (Change the port number to match whatever you used as your SSH port).

Save the file and restart Fail2Ban to put the new rules into effect:

```
sudo service fail2ban restart
```

Add a firewall

We'll add an iptables [hn.my/iptables] firewall to the server that blocks all incoming and outgoing connections except for ones that we manually approve. This way, only the services we choose can communicate with the internet.

The firewall has no rules yet. Check it out:

```
sudo iptables -L
```

Setup firewall rules in a new file:

```
sudo nano /etc/iptables.firewall.rules
```

The following firewall rules will allow HTTP (80), HTTPS (443), SSH (44444), ping, and some other ports for testing. All other ports will be blocked.

Paste thist gist [gist.github.com/4665695] into `/etc/iptables.firewall.rules`.

Activate the firewall rules now:

```
sudo iptables-restore < /etc/iptables.firewall.
rules
```

Verify that the rules were installed correctly:

```
sudo iptables -L
```

Activate the firewall rules on startup:

```
sudo nano /etc/network/if-pre-up.d/firewall
```

Paste this into the `/etc/network/if-pre-up.d/firewall` file:

```
#!/bin/sh
/sbin/iptables-restore < /etc/iptables.firewall.
rules
```

Set the script permissions:

```
sudo chmod +x /etc/network/if-pre-up.d/firewall
```

Get an email anytime a user uses sudo

I like to get an email anytime someone uses sudo. This way, I have a "paper trail" of sorts, in case anything bad happens to my server. I use a Gmail filter to file these away and only look at them occasionally.

Create a new file for the sudo settings:

```
sudo nano /etc/sudoers.d/my_sudoers
```

Add this to the file:

```
Defaults    mail_always
Defaults    mailto="feross@feross.org"
```

Set permissions on the file:

```
sudo chmod 0440 /etc/sudoers.d/my_sudoers
```

This isn't mentioned anywhere on the web, as far as I know, but in order for the "mail on sudo use" feature to work, you need to install an MTA server. `sendmail` is a good choice:

```
sudo aptitude install sendmail
```

Now, you should get an email anytime someone uses `sudo`!

Improve Server Stability

VPS servers can easily run out of memory during traffic spikes.

For example, most people don't change Apache's default setting which allows 150 clients to connect simultaneously. This is way too large a number for a typical VPS server. Let's do the math. Apache's processes are typically ~25MB each. If our website gets a temporary traffic spike and 150 processes launch, we'll need 3750MB of memory on our server. If we don't have this much (and we don't!), then the OS will grind to a halt as it swaps memory to disk to make room for new processes, but then immediately swaps the stuff on disk back into memory.

No useful work gets done once swapping happens. The server can be stuck in this state for hours, even after the traffic rush has subsided. During this time, very few web requests will get serviced.

It's very important to configure your applications so memory swapping does not occur. If you use Apache, you should set `MaxClients` to something more reasonable like 20 or 30. There are many other optimizations to make, too.

Reboot server on out-of-memory condition

Still, in cases where something goes awry, it is good to automatically reboot your server when it runs out of memory. This will cause a minute or two of downtime, but it's better than languishing in the swapping state for potentially hours or days.

You can leverage a couple kernel settings and Lassie to make this happen on Linode.

Adding the following two lines to your `/etc/sysctl.conf` will cause it to reboot after running out of memory:

```
vm.panic_on_oom=1
kernel.panic=10
```

The `vm.panic_on_oom=1` line enables panic on OOM; the `kernel.panic=10` line tells the kernel to reboot ten seconds after panicking.

Miscellaneous nice-to-haves

These next things are not required (in fact, nothing in this guide really is), but are nice to do.

Set up reverse DNS

The reverse DNS system allows one to determine the domain name that lives at a given IP address. This is useful for network troubleshooting — (ping, traceroute, etc.), as well as email anti-spam measures.

It's pretty easy to set up. From the Linode Manager, select your Linode, click on "Remote Access", then click on "Reverse DNS" (under "Public IPs"). Type in your domain, and that's it!

Set up a private IP address

Private IPs are useful for communicating data on the Linode network, i.e. Linode to Linode. This is handy if you have multiple Linodes (say, one for your web server and one for your database). Private network traffic is more secure (only other Linode customers can see it, vs. the whole internet), faster (the traffic never has to leave the datacenter if both Linodes are in the same datacenter), and free (doesn't count towards your monthly bandwidth quota).

I currently put my database server on its own Linode, so that I can scale it independently of my frontend servers and debug performance issues easier since the systems are isolated. This hasn't been super-handy yet, but if one of my sites gets a huge traffic rush, I bet it will be immensely useful.

It's easy to set up. On the Remote Access tab, click Add a Private IP.

Then, edit the file `/etc/network/interfaces` to contain:

```
# The loopback interface
auto lo
iface lo inet loopback

# Configuration for eth0 and aliases

# This line ensures that the interface will be
# brought up during boot.
auto eth0 eth0:0
# eth0 - This is the main IP address that will
# be used for most outbound connections.
```

```
# The address, netmask and gateway are all
# necessary.
iface eth0 inet static
 address 12.34.56.78
 netmask 255.255.255.0
 gateway 12.34.56.1

# eth0:0 - Private IPs have no gateway (they are
# not publicly routable) so all you need to
# specify is the address and netmask.
iface eth0:0 inet static
 address 192.168.133.234
 netmask 255.255.128.0
```

Of course, adjust the IP addresses to reflect your own addresses from the Remote access tab.

Then, restart your Linode and remove DHCP since we're using static networking now:

```
sudo aptitude remove isc-dhcp-client dhcp3-cli-
ent dhcpcd
```

Install Useful Server Software

At this point, you have a pretty nice server setup. Congrats! But your server still doesn't do anything useful. Let's install some software.

Install a compiler

A compiler is often required to install Python packages and other software, so let's just install one up-front.

```
sudo aptitude install build-essential
```

Install MySQL

```
sudo aptitude install mysql-server
libmysqlclient-dev
```

Set root password when prompt asks you.

Verify that MySQL is running.

```
sudo netstat -tap | grep mysql
```

For connecting to MySQL, instead of the usual PHPMyAdmin, I now use Sequel Pro [sequelpro.com], a free app for Mac.

Improve MySQL security

Before using MySQL in production, you'll want to improve your MySQL installation security. Run:

```
mysql_secure_installation
```

This will help you set a password for the root account, remove anonymous-user accounts, and remove the test database.

Keep your MySQL tables in tip-top shape

Over time your MySQL tables will get fragmented and queries will take longer to complete. You can keep your tables in top shape by regularly running `OPTIMIZE TABLE` on all your tables. But, since you'll never remember to do this regularly, we should set up a cron job to do this.

Open up your crontab file:

```
crontab -e
```

Then, add the following line:

```
@weekly mysqlcheck -o --user=root
--password=<your password here> -A
```

Also, you can try manually running the above command to verify that it works correctly.

Backup your MySQL databases

The excellent automysqlbackup utility can automatically make daily, weekly, and monthly backups of your MySQL database.

Install it:

```
sudo aptitude install automysqlbackup
```

Now, let's configure it. Open the configuration file:

```
sudo nano /etc/default/automysqlbackup
```

By default, your database backups get stored in `/var/lib/automysqlbackup` which isn't very intuitive. I recommend changing it to a folder within your home directory. To do this, find the line that begins with `BACKUPDIR=` and change it to `BACKUPDIR="/home/<your username>/backups/"`

You also want to get an email if an error occurs, so you'll know if automatic backups stop working for some reason. Find the line that begins with `MAILADDR=` and change it to `MAILADDR="<your email address>"`.

Close and save the file. That's it!

Install Python

Install Python environment:

```
sudo aptitude install python-pip python-dev
sudo pip install virtualenv
```

This creates a global "pip" command to install Python packages. Don't use it, because packages will be installed globally. Instead, use virtualenv.

Create a new virtualenv Python environment with:

```
virtualenv --distribute <environment_name>
```

Switch to the new environment with:

```
cd <environment_name>
source bin/activate
```

Note that the name of your environment is added to your command prompt.

Install Python packages with "pip" inside of virtualenv:

```
pip search <package_name>
pip install <package_name>
```

This is the best Python workflow that I've found. Let me know if you know of a better way to manage Python packages and Python installations.

Install Nginx

```
sudo aptitude install nginx
```

Install Apache

```
sudo aptitude install apache2
```

Install PHP5

```
sudo aptitude install php5 libapache2-mod-php5
php5-mysql
sudo service apache2 restart
```

Install Node.js

```
sudo aptitude install python-software-properties
sudo add-apt-repository ppa:chris-lea/node.js
sudo aptitude update
sudo aptitude install nodejs npm nodejs-dev
```

Install MongoDB

Follow instructions on 10gen's site: Install MongoDB on Ubuntu. [hn.my/instmongo]

Install Redis

```
sudo aptitude install redis-server
```

Setup Automatic Backups

Backups are really important. Linode offers a paid backup service that's really convenient if you accidentally destroy something and need to restore your Linode quickly. It's $5 per month for the smallest Linode. I enable it on all my Linodes.

If you want even more peace of mind (or don't want to pay for Linode's backup service), you can roll your own simple backup solution using `rsync`.

You will need access to another Linux server (maybe another Linode?) or a home server. I just installed Ubuntu on an old desktop computer to use as a backup server.

We're going to create a weekly cronjob that backs up our Linode's home directory to a backup server. I keep all the files that I would want to backup in my home folder, so this works for me.

Open your crontab:

```
crontab -e
```

Add this line to the file:

```
@weekly rsync -r -a -e "ssh -l <your username
on backup server> -p <ssh port number of backup
server>" --delete /home/<your username> <host-
name or ip address of backup server>:/path/to/
some/directory/on/backup/server
```

I recommend running the above command manually to make sure you have it right before adding it to your crontab file.

That's it! Happy hacking! ■

Feross Aboukhadijeh is a 22-year old Stanford CS student/teacher, web developer, designer, and security researcher. He is the founder of StudyNotes [studynotes.org] where he is helping students to learn faster and study better.

Reprinted with permission of the original author.
First appeared in *hn.my/linode* (feross.org)

Simplify Your Life With an SSH Config File

By JOEL PERRAS

If you're anything like me, you probably log in and out of a half dozen remote servers (or these days, local virtual machines) on a daily basis. And if you're even more like me, you have trouble remembering all of the various usernames, remote addresses and command line options for specifying such things as a non-standard connection port or a local port to forward to a remote server.

Luckily, there are a few ways that we can simplify these tedious, repetitive actions.

Shell Aliases

Let's say that you have a remote server named `dev.example.com`, which has not been set up with public/private keys for password-less logins. The username to the remote account is `fooey`, and to reduce the number of scripted login intrusion attempts, you've decided to obfuscate the default SSH port to 2200 from the normal default of 22. This means that a typical login command would look like:

```
$ ssh fooey@dev.example.com -p 2200
password: *************
```

Not horribly complex or long, but still cumbersome to type out a dozen times a day.

We can make things simpler and more secure by using a public/private key pair:

```
$ # Assuming your keys are properly setup...
$ ssh fooey@dev.example.com -p 2200
```

Note: I highly recommend using `ssh-copy-id` *for moving your public keys around. It will save you quite a few folder/file permission headaches.*

Now, this doesn't seem all that bad. To cut down on the verbosity you could also create a shell alias:

```
$ alias sshdev='ssh fooey@dev.example.com -p
2200'
$ # To connect:
$ sshdev
```

This works surprisingly well, and can scale linearly for every new server you need to work with: Just add an additional alias to your `.bashrc` or `.zshrc`, and voilà.

~/.ssh/config

Even with the simplicity of the method described previously, there's a much more elegant and flexible solution to this problem. Enter the SSH config file:

```
# contents of $HOME/.ssh/config
Host dev
    HostName dev.example.com
    Port 22000
    User fooey
```

This means that I can simply issue `$ ssh dev` in my terminal and the options will be read from the configuration file automatically, and on every invocation. Easy peasy.

Let's see what else we can do with just a few simple configuration directives.

Personally, I manage a few public/private keypairs due to having multiple machines (work/home/laptop). Say, for illustrative purposes only, that I have a key that I use uniquely for my github account. Let's set it up so that that particular private key is used for all my github-related operations:

```
# contents of $HOME/.ssh/config
Host github.com
    IdentityFile ~/.ssh/github.key
```

The use of `IdentityFile` allows me to specify exactly which private key I wish to use for authentification with the given host instead of specifying this as a command line parameter:

```
$ ssh -i ~/.ssh/blah.key username@host.com
```

However, the use of a config file with the `IdentityFile` directive is pretty much your only option if you want to specify which identity to use for any git commands. This also opens up the very interesting concept of further segmenting your keypairs:

```
Host github-work
    User git
    HostName github.com
    IdentityFile ~/.ssh/github.work.key

Host github-home
    User git
    HostName github.com
    IdentityFile ~/.ssh/github.home.key

Host github-laptop
    User git
    IdentityFile ~/.ssh/github.laptop.key
```

Which means that if I want to clone a repository using my work credentials, I can simply use the following:

```
$ git clone git@github-work:orgname/some_repository.git
```

Going further

As a security-conscious developer, I make sure to set up firewalls on all of my servers and make them as restrictive as possible. In many cases, this means that the only ports that I leave open are `80/443` (for webservers) and `22` for SSH.

On the surface, this seems to prevent me from using things like a desktop MySQL GUI client, which expects port `3306` to be open and accessible on the remote server you are connecting to. The informed reader will note, however, that a simple local port forward can save you:

```
$ ssh -f -N -L 9906:127.0.0.1:3306 coolio@database.example.com
$ # -f puts ssh in background
$ # -N makes it not execute a remote command
```

This will forward all local port `9906` traffic to port `3306` on the remote `dev.example.com` server, letting me point my desktop GUI to localhost (`127.0.0.1:9906`) and have it behave exactly as if I had exposed port `3306` on the remote server and connected directly to it.

Now I don't know about you, but remembering that sequence of flags and options for SSH can be a complete pain. Luckily, our config file can help alleviate that:

```
Host tunnel
    HostName database.example.com
    IdentityFile ~/.ssh/coolio.example.key
    LocalForward 9906 127.0.0.1:3306
    User coolio
```

This means I can simply do:

```
$ ssh -f -N tunnel
```

And my local port forwarding will be enabled using all of the configuration directives I set up for the tunnel host. Slick.

Homework

There are quite a few configuration options that you can specify in `~/.ssh/config`, and I highly suggest consulting the online documentation or the `ssh_config` man page [hn.my/sshconfig]. Some interesting/useful things that you can do include:

- Changing the default number of connection attempts
- Specifying local environment variables to be passed to the remote server upon connection
- Using of `*` and `?` wildcards for matching hosts.

And much, much more — the `ssh_config` man page is 747 lines long, and consists almost entirely of configuration file directives and short explanations of what those directives do. Take a look! I'm certain you'll be surprised at what you find. ■

Joel Perras is a physicist turned Big Data geek. He is a partner at Fictive Kin, where he gets to build applications to change the way the world use the web.

Reprinted with permission of the original author.
First appeared in *hn.my/simplify* (nerderati.com)

What I've Learned About Learning

By REGINALD BRAITHWAITE

I HAVE A RATHER glaring life-long weakness, a behavior that has tripped me up many times. You would think that I would have noticed it and corrected my behavior in my teens or twenties, but no, it has persisted. While I am much better at correcting myself, it is extremely persistent and requires constant vigilance to suppress.

The behavior in question is this: when I am learning something new, I suffer from laziness, impatience, and hubris. I try to grasp the gist of the thing, the conclusion, and then I stop. I figure I "understand" it, so I must be done learning.

This is wrong for me. I am blessed with a quick mind for certain subjects, so there are times when I am reading something, or someone is explaining something, and I can work out the obvious implications. Someone is telling me about aspect-oriented programming, and I start thinking about cross-cutting concerns like authorization. Or perhaps database access. Then I ask myself whether AOP is related to the "Unobtrusive JavaScript" style. Or if it's really fine-grained dependency injection.

I'm impatient to learn, so I try to jump to the end. I don't "do the work" of taking it step by step, doing exercises with the material, building my knowledge like a pyramid with a broad foundation. If a technology seems interesting, I want to jump right into the deep end and try it on an important project instead of researching it more thoroughly or playing with it a bit on side-projects.

This has been wrong more often than not.

Every idea has "big implications." Decoupling. Refactoring. Events. Idempotence. Whatever. But ideas in execution have many, many little implications, little caveats and gotchas, rough edges and leaky abstractions. These "little ideas" are less important than the big ideas in theory, but in practice each failure to grasp an implication or consequence leads almost directly to a flaw in the finished work.

The wrongness of my laziness, impatience, and hubris is apparent. When I think that I "grasp" an idea, I'm really only grasping the big idea. Or what I think is the big idea in my hubris. I'm assuming that what I don't know about the idea can't hurt me. But what I don't know about an idea can hurt me and often has.

I don't always make this mistake. Sometimes an idea catches my fancy and I find myself playing with it. Combinators grabbed me in this way. I was fascinated by the book To Mock a Mockingbird many years ago, and when I started Ruby programming, I had a chance to try these ideas out in practice.

I worked from Ruby to Combinatorial Logic and from Combinatorial Logic to Ruby at the same time. I tried to view certain meta-programming ideas from the perspective of CL, to fit them within the framework. This opened up some insights that had eluded me when I thought that I had "grasped" Ruby meta-programming. And when I took some of the combinators and tried to find practical uses for them in Ruby, I learned some more.

It turns out, there really is no substitute for experience with an idea. Experience that is obtained through practice, through repeated application of principles to problems, not just from skimming a text.

A decade ago, I would have read an article like this and summarized it thusly: "There's more to an idea than the obvious implication, there are some details you need to learn as well." I'd have been wrong. There's an important factor my mental model of ideas and implications ignores.

> *"A programming language that doesn't affect the way you think about programming is not worth learning." — Alan Perlis*

My personal experience is that "learning" a programming language requires writing programs in that language. If I tell you that Scheme is homoiconic, that it has just five special forms, and that it has hygienic macros do you "know" Scheme? Can you say that it "affects the way you think about programming"?

From what I wrote above, we say "No, because there are many implications of these three features of the language that are not obvious, that require further study." But I think there's something else.

Knowing how to do something is not the same thing as doing that thing. When you actually do the thing, and when you incorporate it into your life, it becomes a mental habit, it becomes part of who you are and how you think.

That is when a language affects the way you think about programs: when its ideas become part of your mental habits. And a language's ideas only become your mental habits when you program in that language on a regular basis. This doesn't happen from playing with it a bit here and a bit there. It doesn't happen from reading a lot of books and blog posts. It doesn't happen when you snap your fingers and think you "get" its big ideas.

You have to go beyond thinking you know how, you have to go beyond actually knowing how, you have to go out and do it. Again and again and again until it becomes a habit. I think this is true of much more than just programming languages. Everything I've learned works the same way: There is a difference between knowing how to do it and doing it enough to change your way of thinking.

> *In cognitive therapy, you have to "do the work," you have to grind it out and do the exercises day in and day out. Week after week. Month after month.*

I've experienced this in a very direct way with Cognitive Therapy. I can tell you (and I did) that one of the ways to combat depression is to change the way you explain negative events in your life, to view them as being impersonal or external, specific, and temporary. I can tell you that you should view positive events as personal or internal, general, and permanent. And I know you will snap your fingers and intuit how this can change your moods and outlook.

But knowing how to change your moods is not the same thing as changing your moods. In cognitive therapy, you have to "do the work," you have to grind it out and do the exercises day in and day out. Week after week. Month after month. It's the doing of cognitive therapy that changes your moods, not the grasping of its big principles nor the acquisition of the little implications. Just doing it. Day after day after day.

Reasoning by analogy is notoriously unreliable, but there seems to be a deep truth here about the business of learning ideas. There are the big implications I can grasp, sometimes quickly. But there are also the "little" implications that require practice and experimentation, and when I am impatient and ignore them, I suffer.

And finally there is the way that an idea affects my thinking which comes only from sustained effort applying the idea. Picking it up and playing with it isn't enough, I need to use it every day if I want it to change me in any serious way.

My weakness is thinking that when I first grasp an idea, I think I'm done. I've gotten better over time. I have learned to exercise ideas, to write and use little libraries, even to write essays like this to help me think the little implications through.

But I mustn't be fooled into thinking I'm done. So if you'll excuse me, I'm off to do the work. ■

Reginald Braithwaite is a software developer at Leanpub, where he and his colleagues take the friction out of writing and selling books. He has more than twenty years of hands-on experience creating software products and leading software teams. He currently applies extremely deep Ruby, JavaScript, CoffeeScript, and advanced programming expertise to crafting well-factored, maintainable code.

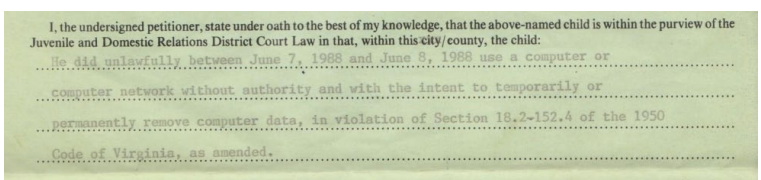
Reprinted with permission of the original author. First appeared in *hn.my/learned* (raganwald.posterous.com)

I Was a Teenage Hacker

By JEFF ATWOOD

TWENTY-FOUR YEARS AGO today, I had a very bad day.

On August 8, 1988, I was a senior in high school. I was working my after school and weekend job at Safeway as a cashier, when the store manager suddenly walked over and said I better stop ringing up customers and talk to my mother on the store phone right now. Mom told me to come home immediately because, well, there were police at the front door asking for me with some legal papers in hand.



I, the undersigned petitioner, state under oath to the best of my knowledge, that the above-named child is within the purview of the Juvenile and Domestic Relations District Court Law in that, within this city/county, the child:

He did unlawfully between June 7, 1988 and June 8, 1988 use a computer or computer network without authority and with the intent to temporarily or permanently remove computer data, in violation of Section 18.2-152.4 of the 1950 Code of Virginia, as amended.

Like I said, definitely not a good day. The only sliver of good news was that I was still 17 at the time, so I enjoyed the many protections that the law provides to a minor. Which I shall now throw away by informing the world that I am a dirty, filthy, reprehensible adult criminal. Thanks, law!

One of the problems you had in the pre-Internet 1980s as a hard-core computer geek was that all the best bulletin boards and online services were kind of expensive. Either because you had to pay an hourly fee to access them, like CompuServe, or because they were a long distance modem call. Or both. Even after the 1984 AT&T breakup, long distance at around 20-30 cents a minute was a far, far cry from today's rates. (Does anyone actually even worry about how much voice calls cost any more, to anywhere in the world? This, my friends, is progress.)

Remember, too, that this is back when 9600 baud modems were blazing, state of the art devices. For perspective, the ultra-low-power wireless Bluetooth on your phone is about 80 times faster. If you wanted to upload or download any warez software, that meant potentially hours on your modem at rates of around $20/hour. Adjusted for inflation, that's closer to $40 in 2012 dollars. My family wasn't well off enough to afford a second telephone line, so most of my calling was done late at night both because the rates were lower, and also so that I wouldn't be monopolizing the telephone. Nothing was worse than the dreaded "mom picked up the phone" disconnect to an elite difficult-to-access BBS with limited slots.

One way or another, I eventually got involved with the seedier side of the community, even joining a lesser Apple // pirate group. Probably my main claim to fame is that while trolling BBSes, I personally discovered and recruited a guy who turned out to be an amazing cracker. He was so good he eventually got recruited away.

> **“Obtaining access to free, unlimited long distance calling rapidly became an urgent priority in my teenage life.”**

I was, at best, a footnote to a footnote to a footnote in Apple // history. This was mainly a process of self-discovery for me. I learned I was the type of geek who doesn't even bother attending his high school prom, partially because I was still afraid of girls even as a high school senior, yes, but mainly because I was so addicted to computers and playing my tiny role in these nascent online communities. I was, and am, OK with that. This is the circuitous path of 30 years that led me to create Stack Overflow. And there's more, so much more, but I can't talk about it yet.

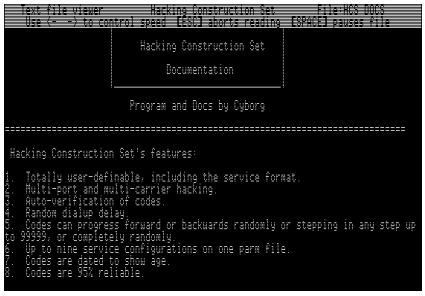
But addicted, I think, is too weak a word for what I felt about being a part of these oddball, early online home computer communities. It was more like an all-consuming maniacal blood lust. So obtaining access to free, unlimited long distance calling rapidly became an urgent priority in my teenage life. I needed it. I needed it so bad. I had to have it to talk on the phone to the other members of my motley little crew, who were spread all over the USA, as well as for calling BBSes.

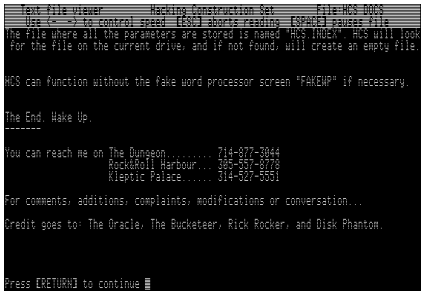
I can't remember exactly how I found it, probably on one of the BBSes, but I eventually discovered a local 804 area code number for “calling cards” that accepted a 5 digit PIN, entered via touch-tone phone. Try over and over, and you might find some valid PIN codes that let you attain the holy grail of free long distance calling. Only one small problem: it's a crime. But, at least to my addled teenage brain, this was a victimless crime, one that I had to commit. The spice must flow!

All I had to do is write software to tell the modem to dial over and over and try different combinations. Because I was a self-taught programmer, this was no problem. But because I was an overachieving self-taught programmer, I didn't just write a program. No, I went off and built a full-blown toolkit in AppleBasic, with complete documentation and the best possible text user interface I could muster, and then uploaded it to my favorite BBSes so every other addict could get their online modem fix, too. I called it The Hacking Construction Set, and I spent months building it. I didn't just gold plate; I platinum plated this freaking thing, man. (Yes, I know the name isn't really correct. I read as many 2600 textfiles as the next guy. This is mere phreaking, not hacking, but I guess I was shooting for poetic license. Maybe you could use the long distance dialing codes to actually hack remote machines or something.)

I never knew if anyone else ever used my little program to dial for calling codes. It certainly worked for me, and I tried my level best to make it work for all the possible dialing situations I could think of. It even had an intro screen with music and graphics of my own creation. But searching now, for the first time in 24 years, I found my old Hacking Construction Set disk image on an Apple ROM site [hn.my/hcs]. It even has real saved numbers in the dialing list! Someone was using my illicit software!

If you're curious, fire up your favorite Apple // emulator and give the disk image [hn.my/dsk] a spin. Don't forget to connect your modem. There's full blown documentation accessible from the main menu. Which, re-reading now, was actually not half bad, if I do say so myself:





I used to regularly call BBSes in Florida, California, and Missouri? That's news to me; I haven't seen any of this stuff in over 24 years! All I did was upload a disk image to a few BBSes in 1986. After all that time, to discover that someone used and loved my little bit of software still gives me a little thrill. What higher praise is there for a software developer?

About that trouble. Using my own software got me in trouble with the law. And deservedly so; what I wrote the software to do was illegal. I hired a local lawyer (who, as I recall, was missing a hand; he had a prosthetic hand that was almost impossible not to look at) to represent me. It was quite clear at preliminary hearings that the Chesterfield County court system did not see any computer crime cases, and they had absolutely no idea what to make of me, or what this was all about. All they saw was a smart kid with a bit of bad judgment who loved computers and was headed to the University of Virginia, most likely not a life as a career criminal. So the case was dismissed for the cost of lawyer's fees. Which, for the record, I had to pay myself, using my income as a Safeway cashier.

This was definitely a wake up call for me. In the summer of 1988, I was about to graduate from high school, and I thought I'd try being just a regular guy at college, with less of an obsessive focus on computers that causes me to get in trouble with the law, and perhaps spread my wings to other interests. Who knows, maybe even girls!

That didn't last long. Because after all these years, I must confess I've grown to love my own bad judgment. It's led me to the most fascinating places. ■

Jeff Atwood lives in Berkeley, CA with his wife, two cats, and a whole lot of computers. He is best known as the author of popular blog Coding Horror and the cofounder of Stack Overflow with Joel Spolsky.

Reprinted with permission of the original author.
First appeared in *hn.my/teenhack* (codinghorror.com)

stripe

Accept payments online.

MEMSET® HOSTING

Rent your IT infrastructure from Memset and discover the incredible benefits of cloud computing.

MINISERVER™ CLOUD COMPUTE

From £0.015p/hour
to 4 x 2.9 GHz Xeon cores
31 GBytes RAM
2.5TB RAID(1) disk

MEMSTORE™ CLOUD STORAGE

£0.07p/GByte/month or less
99.999999% object durability
99.995% availability guarantee
RESTful API, FTP/SFTP and CDN Service

SCAN THE CODE FOR MORE INFORMATION

MEMSET® HOSTING

CarbonNeutral® hosting

ISO 9001: Quality
ISO 14001: Environmental
ISO 27001: Security

PC PRO Excellence AWARDS 2011 WINNER BEST WEB HOST

Best Managed Service WINNER The ISPAs 2011 www.ispaawards.org.uk

Find out more about us at www.memset.com

or chat to our sales team on 0800 634 9270.